Évaluation d'un jeu sérieux pour l'apprentissage de la programmation

Mathieu Muratet, Patrice Torguet, Fabienne Viallet, Jean Pierre Jessel

Évaluation d'un jeu sérieux pour l'apprentissage de la programmation

Mathieu Muratet[*,***] — **Patrice Torguet**[*,***] — **Fabienne Viallet**[**,***] — **Jean-Pierre Jessel**[*,***]

[*] *IRIT, équipe VORTEX*

{Mathieu.Muratet, Patrice.Torguet, Jean-Pierre.Jessel}@irit.fr

[**] *CREFI-T, équipe DiDiST*

fabienne.viallet@iut-tlse3.fr

[***] *Université Paul Sabatier, 118 Route de Narbonne F-31062 Toulouse cedex 9*

RÉSUMÉ. *Les jeux vidéo font partie de notre culture au même titre que la télévision, les films ou les livres. Les jeux vidéo ayant un but autre que le simple divertissement sont appelés jeux sérieux. Nous émettons l'hypothèse que ce type de jeu constitue un excellent moyen pour motiver les étudiants à programmer. Cet article présente la conception et l'évaluation d'un jeu sérieux destiné à renforcer les compétences des étudiants dans le domaine de la programmation. Ce jeu sérieux est basé sur un type de jeu populaire et apprécié de nos étudiants : un jeu de stratégie temps réel. Suite à l'analyse des compétences requises pour l'apprentissage de la programmation et des caractéristiques des jeux de stratégie temps réel, nous présentons le jeu sérieux et son évaluation.*

ABSTRACT. *Video games are part of our culture like TV, movies and books. Video games with other goals than entertainment are called: Serious Games. We believe this kind of software can be used to increase students' interest in computer science. This paper presents a study around a serious game dedicated to strengthening programming skills. Real-Time Strategy game, which is a popular game genre, seems to be the most suitable kind of game to support such a serious game. From programming teaching features to video game characteristics, we describe the serious game and its evaluation.*

MOTS-CLÉS : *Jeux sérieux, apprentissage de la programmation, évaluation, jeux de stratégie temps réel, Prog&Play.*

KEYWORDS: *Serious games, learning programming, evaluation, real time strategy games, Prog&Play.*

1. Introduction

Depuis la première explosion des jeux vidéo dans les années 80, l'industrie du jeu vidéo a pris une place importante dans l'économie mondiale. D'après les chiffres de l'*Entertainment Software Association* [1], en 2008 le marché du jeu vidéo aux Etats-Unis a atteint 11,5 milliards de dollars et a dépassé le marché du cinéma [2] : 10 milliards de dollars. Les étudiants, actuellement à l'université, sont nés avec les jeux vidéo. Cette composante ludique fait partie de leur culture au même titre que la télévision, les films ou les livres.

Parallèlement au développement des jeux vidéo, les étudiants se détournent des sciences. En informatique, par exemple, d'après Crenshaw *et al.* (2008) et Kelleher (2006), le nombre d'étudiants est en chute libre. De plus, « les établissements d'enseignement supérieur et universités affirment que régulièrement 50% ou plus de leurs étudiants ayant initialement choisi des études en informatique décident rapidement d'abandonner » (ACM/IEEE, 2005, p. 39). Notre université subit le même phénomène avec une baisse de 17,6% d'étudiants inscrits en première année de licence sur les quatre dernières années et une baisse de 30% en deuxième année de licence informatique sur les sept dernières années. Ainsi, la recherche dans le domaine de l'enseignement de l'informatique consacre une part importante de ses travaux aux problèmes du recrutement et du maintien des étudiants dans les formations informatiques (Fincher *et al.*, 2004). Une approche possible consiste à utiliser la culture vidéoludique des étudiants pour les motiver, à travers le jeu, à investir du temps dans la pratique de la programmation.

Nous proposons d'étudier un jeu sérieux pour l'apprentissage de la programmation, le recrutement et le maintien des étudiants dans la filière informatique. Après avoir présenté le concept de jeu sérieux et les caractéristiques liées à l'apprentissage de la programmation, nous évoquons dans cet article plusieurs approches destinées à faciliter l'apprentissage de la programmation (section 2). Suite à cette énumération, nous détaillons la conception d'un jeu sérieux dédié à l'apprentissage de la programmation (section 3). Enfin, nous présentons le protocole itératif d'évaluation ainsi que les résultats obtenus lors des expérimentations (section 4).

2. Jeux sérieux et apprentissage de la programmation

Les jeux sérieux sont présents dans de nombreux secteurs d'activité dont l'éducation. Ces outils issus du domaine de l'industrie du jeu vidéo peuvent-ils être utilisés pour former les étudiants de cette même discipline ? Pour répondre à cette question, il convient de définir le concept de jeu sérieux. Suite à cette définition, il sera possible

1. *Essential facts about the computer and video game industry* : `http://www.theesa.com/facts/pdfs/ESA_EF_2009.pdf` consulté le 8 Janvier 2010

2. *US Movie Market Summary for 2008* : `http://www.the-numbers.com/market/2008.php` consulté le 8 Janvier 2010

d'identifier parmi les approches existantes dédiées à l'apprentissage de la programmation, celles en rapport avec les jeux sérieux.

2.1. *Jeux sérieux*

2.1.1. *Définition*

Le terme « jeu sérieux », tel que nous l'entendons aujourd'hui, semble avoir été évoqué pour la première fois, en 1970, par Clark Abt dans son livre *Serious Games* (Abt, 1970). Dans cet ouvrage, le terme *Serious Games* est utilisé pour des jeux de cartes ou de plateau conçus dans un but éducatif.

Actuellement, l'expression « jeu sérieux » est généralement utilisée dans un contexte informatique plus prononcé et caractérise une gamme de jeux vidéo bien plus large que le simple jeu éducatif. En effet, un jeu vidéo conçu avec un objectif autre que le simple divertissement peut souvent être considéré comme un jeu sérieux. Zyda (2005) caractérise le « jeu sérieux » comme « *un défi intellectuel lancé sur un ordinateur selon des règles spécifiques. Il utilise le divertissement pour servir à la formation, à l'éducation, à la santé, à la sécurité civile et comme stratégie de communication dans des milieux institutionnels ou privés* ». Outre cette définition, Zyda identifie un point critique de la conception d'un jeu sérieux concernant le positionnement du « jeu » par rapport au concept de « sérieux » (message véhiculé qu'il soit d'ordre formatif, éducatif, informatif, etc.). Zyda (Zyda, 2005) écrit à ce propos que « *la pédagogie doit être subordonnée au scénario du jeu - la composante ludique doit primer* ». Il convient toutefois de s'assurer que ces deux composantes concordent pour obtenir un jeu sérieux cohérent et éventuellement pertinent (Michaud *et al.*, 2008).

2.1.2. *Exemples de jeux sérieux*

La définition de Zyda présentée dans la partie précédente met en exergue la diversité des domaines d'application dans lesquels les jeux sérieux peuvent être employés. Pour illustrer cette hétérogénéité, nous présentons par ordre chronologique trois exemples de jeux sérieux aux domaines d'application et aux objectifs différents : « *America's Army* » [3], « *Darfur is Dying* » [4] et « *Moonshield* » [5].

Le jeu « *America's Army* » a été initialement développé comme outil de recrutement pour l'armée des États-Unis. Il est devenu l'un des premiers jeux sérieux à remporter un réel succès et est actuellement l'un des dix jeux d'action les plus populaires joués en ligne. « *America's Army* » est un jeu complexe de tir à la première personne, multijoueur et gratuit.

« *Darfur is Dying* » a pour but de faire prendre conscience au grand public des conséquences de la crise du Darfour sur la population. Le joueur incarne un réfugié

3. `http://www.americasarmy.com` consulté le 8 Janvier 2010
4. `http://www.darfurisdying.com` consulté le 8 Janvier 2010
5. `http://www.moonshield.com/game/play` consulté le 25 Février 2010

qui doit aller puiser de l'eau pour aider à développer son camp tout en évitant des bandits. C'est un mini jeu gratuit et accessible en ligne.

« *Moonshield* » est un jeu de gestion/stratégie. Il propose au joueur, dans un contexte d'anticipation proche, de mettre en œuvre les technologies de la société Thales pour préserver la terre d'une pluie d'astéroïdes qui pourrait détruire toute civilisation. Il a été construit afin d'attirer les jeunes ingénieurs diplômés à soumettre leur candidature auprès de cette entreprise. A priori c'est un succès : 50 000 personnes sont allées du jeu vers le site de recrutement sur 80 000 joueurs différents. C'est un jeu gratuit et accessible en ligne.

Ces quelques jeux sérieux utilisent le divertissement pour atteindre différents objectifs : « *America's Army* » est un **outil de recrutement** ; « *Darfur is dying* » tente de **sensibiliser** ; « *Moonshield* » est utilisé à des fins de **communication**. Qu'en est-il d'un jeu sérieux dédié à l'enseignement de la programmation ? Existe-t-il un besoin pour ce type d'outil ?

2.2. *Apprentissage des fondamentaux de la programmation*

Pour déterminer la pertinence d'un jeu sérieux pour l'enseignement de la programmation, il convient de présenter les caractéristiques liées à la discipline, de détailler les problèmes rencontrés par les étudiants et d'exposer quelques solutions issues de la littérature.

2.2.1. *Caractéristiques de l'informatique*

Le rapport de l'ACM/IEEE (2005) décrit le panorama des différentes approches de l'enseignement de l'informatique à l'université. Ce rapport divise l'informatique en cinq disciplines majeures : l'ingénierie de l'informatique (*Computer Engineering*), l'informatique scientifique (*Computer Science*), les systèmes d'information (*Information Systems*), les technologies de l'information (*Information Technology*), et le génie logiciel (*Software Engineering*). Ce rapport identifie deux composantes essentielles et communes à ces cinq disciplines : **acquérir les fondamentaux de la programmation** et être capable de **réaliser de petits programmes**. Nous remarquons, qu'aucune recommandation n'est effectuée sur un langage de programmation à utiliser.

2.2.2. *Problèmes rencontrés*

Pour différentes raisons, l'apprentissage des fondamentaux de la programmation est une tâche difficile, en particulier pour les novices.

Premièrement, les étudiants se trouvent confrontés à plusieurs obstacles épistémologiques comme la manipulation des boucles (Ginat, 2004) (Soloway *et al.*, 1983), des conditions, ou l'assemblage de programmes à base de composants. Les structures de contrôle et les algorithmes posent souvent problème de par leur abstraction et leur nature dynamique (Seppälä *et al.*, 2006). Ce phénomène se traduit chez les étudiants

par un manque de compétences en programmation en fin de première année d'étude en informatique (Fincher *et al.*, 2004).

Deuxièmement, l'environnement informatique utilisé tous les jours par les étudiants, pour jouer ou pour chatter par exemple, est très différent des dispositifs déployés dans les enseignements. Ainsi, ils ne voient pas immédiatement les liens entre les deux univers : « Les personnes étudiant les techniques pédagogiques s'accordent à dire que les étudiants novices en informatique trouvent souvent la discipline théorique, technique, ou ennuyeuse » (Stamm, 2007).

Troisièmement, apprendre à programmer suppose d'assister à des cours magistraux, des travaux dirigés et des travaux pratiques. Pour pouvoir programmer, les étudiants doivent connaître les concepts et connaissances liées à la programmation, mais pour les apprendre, ils doivent pratiquer la programmation. Pour supplanter ce paradoxe, Greitzer *et al.* (2007) expliquent en particulier qu'une approche efficace consiste à encourager l'apprenant à travailler immédiatement sur des tâches réalistes et significatives.

2.2.3. *Quelques solutions*

De nombreuses réponses sont proposées dans la littérature pour aider les étudiants à dépasser les difficultés présentées dans la section 2.2.2. Une première réponse, présentée par Stevenson *et al.* (2006), consiste à analyser les manuels d'exercices et leurs usages historiques. Les auteurs énoncent alors huit principes de base en vue de concevoir de « bons » exercices de programmation : être basé sur un problème du monde réel ; permettre aux étudiants de réaliser une solution réaliste à ce problème ; leur permettre de se concentrer sur un problème actuel du cours dans un contexte de programmes conséquents ; être stimulant ; être intéressant ; utiliser une ou plusieurs interfaces de programmation applicatives ; proposer plusieurs niveaux d'exercices qui permettent une optimisation des programmes ; permettre la créativité et l'innovation. Les auteurs mettent alors en œuvre leurs huit principes à travers une pédagogie par projet basée sur un robot d'indexation et un évaluateur de courriels indésirables.

Une autre réponse consiste à analyser la manière dont les étudiants perçoivent l'informatique et les raisons pour lesquelles ils abandonnent leur formation. Un manque de sens et de pertinence semble contribuer à la désaffection de cette discipline (Fincher *et al.*, 2004). Pour répondre à ce problème, deux approches sont étudiées : construire des environnements de programmation pour novices ou utiliser des jeux vidéo.

De nombreux environnements de programmation pour novices ont été développés. La plupart utilisent des langages de programmation à base de blocs. Cette métaphore de programmation permet aux étudiants d'oublier la syntaxe, souvent source d'erreurs, pour directement s'intéresser à la résolution de problèmes. En voici quelques exemples : Scratch (Maloney *et al.*, 2004) ; StarLogo *The Next Generation* (Klopfer *et al.*, 2005) ; Alice2 (Kelleher *et al.*, 2002) et Kodu [6].

6. `http://research.microsoft.com/en-us/projects/kodu` consulté le 8 Janvier 2010

L'approche basée sur l'utilisation des jeux vidéo a pour objectif d'accrocher le joueur et de l'amener à programmer. Deux orientations ont été explorées : développer un nouveau jeu vidéo ou jouer à un jeu vidéo. Pour la première orientation, nous pouvons citer trois exemples. Chen *et al.* (2007) proposent de demander aux étudiants, à travers un projet collaboratif, d'implémenter en C++ un mini jeu vidéo interactif en un semestre, à l'aide d'un cadre méthodologique. Gestwicki *et al.* (2008) ont réalisé une étude de cas sur le jeu EEClone. Ce jeu de type arcade est implémenté en Java : les étudiants analysent différents patrons de conception avec EEClone et à partir de cette expérience, étudient comment ils peuvent les intégrer dans leur propre jeu. Leutenegger *et al.* (2007) utilisent le développement de jeux 2D comme thème central et soutiennent que la programmation de jeux vidéo motive surtout les novices.

La deuxième orientation consiste à permettre à l'étudiant d'apprendre la programmation dans le contexte d'un jeu vidéo. La majorité des jeux existants de ce type permettent au joueur de programmer un robot afin de le faire combattre dans une arène contre un ou plusieurs robots programmés par d'autres joueurs. La bataille de robot est exécutée en temps réel sur l'écran de l'ordinateur. Ces jeux sont conçus pour aider les étudiants à pratiquer un langage de programmation spécifique. Ces jeux sont accessibles à tout type de programmeur, des débutants (un simple robot peut être écrit en quelques minutes) aux experts (réaliser une véritable IA - Intelligence Artificielle - peut prendre plusieurs mois). À titre d'exemple, « *Robocode* » (Hartness, 2004) est compatible avec Java, « *Marvin's Arena* »[7] est compatible avec tous les langages .NET, « *Gun-Tactyx* »[8] utilise le langage SMALL et « *Robot Battle* »[9] utilise un langage de script spécifique.

Toujours dans cette seconde orientation, *Colobot*[10] place le joueur dans le rôle d'un explorateur spatial dont l'objectif est d'explorer et coloniser des planètes à la recherche de matières premières nécessaires à sa survie. Le joueur peut progressivement construire et programmer de nouveaux robots pour l'aider dans ces différentes tâches et se protéger contre les créatures primitives et hostiles présentes sur certaines planètes. Ce jeu mise sur un scénario original pour motiver le joueur à résoudre les problèmes auxquels il se trouve confronté. Le langage de programmation utilisé est un langage orienté objet similaire au C++.

2.3. *Conclusion*

Le jeu vidéo peut donc être utilisé de deux manières pour stimuler les étudiants à pratiquer la programmation. La première consiste à leur faire réaliser leur propre jeu

7. `http://www.marvinsarena.com` consulté le 8 Janvier 2010
8. `http://apocalyx.sourceforge.net/guntactyx`consulté le 8 Janvier 2010
9. `http://www.robotbattle.com` consulté le 8 Janvier 2010
10. `http://www.ceebot.com/colobot/index-f.php` consulté le 8 Janvier 2010

vidéo. La deuxième intègre la programmation dans le *gameplay* [11] d'un jeu existant. Dans les deux cas, la programmation est rendue concrète et mise en relation avec le centre d'intérêt des étudiants.

Au regard de la définition des jeux sérieux présentée dans la section 2.1.1, *Colobot* peut être considéré comme un jeu sérieux. C'est le seul exemple de jeu vidéo que nous connaissons qui intègre la programmation dans son scénario de manière interactive et cohérente avec l'univers du jeu. Il inclut également une présentation pédagogique des concepts de programmation via une documentation détaillée consultable directement dans le contexte du jeu.

Les batailles de robots utilisent également l'activité de jeu pour stimuler le joueur à apprendre la programmation. Ils peuvent donc également être considérés comme des jeux sérieux. Contrairement à *Colobot*, ces jeux ne plongent pas le joueur dans une intrigue mais utilisent la compétition pour motiver les étudiants.

Dans tout les cas, ces jeux sérieux ne sont pas utilisables dans différents contextes d'enseignements puisqu'ils reposent sur un langage de programmation spécifique qui peut ne pas correspondre aux choix pédagogiques des enseignants. Or, dans le cadre de notre approche, il nous semble essentiel de construire un outil d'enseignement/apprentissage qui soit adaptable à différents contextes, c'est-à-dire à différents langages, différents paradigmes de programmation, différentes méthodes pédagogiques et différents étudiants. Il devient alors indispensable de construire notre propre jeu sérieux.

3. Développement du jeu sérieux

3.1. *Quel type de jeu utiliser pour supporter notre jeu sérieux ?*

Suite à trois séries d'enquêtes, dont une partie est détaillée par Muratet *et al.* (2009), réalisées de septembre 2007 à juin 2010 auprès de 900 étudiants, il apparaît que les jeux les plus joués par les étudiants interrogés sont les jeux de tir à la première personne, les jeux de rôle, les jeux de stratégie, les jeux de sport et les jeux de course. La classification utilisée ici est celle de la presse spécialisée. Le choix s'est porté vers cette classification en raison de sa visibilité et des moteurs de recherche présents sur les sites Web associés. Ces moteurs de recherche ont été principalement utilisés pour traiter les données récoltées lors des différentes enquêtes. Parmi les cinq familles de jeu les plus jouées, nous choisissons d'utiliser un jeu de stratégie temps réel (sous famille des jeux de stratégie) comme support à notre jeu sérieux car il nous semble être, a priori, le genre de jeu le plus adapté à la pratique de la programmation.

Le choix d'un type de jeu n'est pas suffisant pour entamer la conception d'un jeu sérieux. Il convient avant tout de vérifier la compatibilité entre le jeu support et

11. Le *gameplay* est défini par Johnson *et al.* (2005) comme toutes les activités et stratégies employées par les concepteurs de jeux pour obtenir et garder le joueur engagé et motivé durant tout le jeu.

l'activité de programmation. En effet, cette activité doit s'intégrer au *gameplay* du jeu tout en respectant les règles fondamentales. Dans notre cas, la question est la suivante : les jeux de stratégie temps réel (STR) sont-ils compatibles avec la pratique de la programmation ? Pour répondre à cette question, nous définissons les jeux de STR pour déterminer dans quelle mesure la compatibilité entre ce type de jeu et la programmation peut être satisfaite.

3.1.1. *Définition des jeux de stratégie temps réel*

La présentation de ce type de jeu est réalisée à travers l'analyse du jeu *Dune 2* [12] (un jeu ayant fortement concouru à l'émergence de ce genre). Nous nous appuyons également sur une quarantaine de titres sortis après cet opus pour présenter les évolutions qui ont émergé depuis.

3.1.1.1. Description générale

Pour expliciter le fonctionnement des STR, une analogie avec le jeu d'échec s'avère pertinente. En effet, les échecs font partie des jeux de stratégie, il est donc possible d'effectuer un certain nombre de comparaisons entre les échecs et les STR.

Dans un jeu de STR les parties se réalisent sur un certain nombre de cartes géographiques (équivalentes au plateau du jeu d'échec) avec un ensemble d'unités de départ (équivalentes aux pièces du jeu d'échec). Au même titre que les pièces du jeu d'échec sont déplacées sur le plateau de jeu, les unités d'un jeu de STR évoluent sur une carte qui peut contenir du relief, différents types de sols ainsi que des éléments de décors tels des impacts d'obus, des bâtiments, des ressources, etc. Généralement, les ressources jouent un rôle important dans les jeux de STR, car, présentes en quantités limitées (l'épice pour *Dune 2*), elles sont nécessaires et convoitées par tous les joueurs. Les unités sont dotées également d'un certain nombre d'attributs tels qu'une apparence, une position, un mode de déplacement, mais aussi un capital santé, un champ de vision, un coût, une force, etc. L'ensemble de ces attributs caractérise le type d'une unité.

L'objectif général d'un STR consiste donc à développer une base militaire en vue de lever une armée pour éliminer les troupes et bases ennemies.

3.1.1.2. Règles principales

Dans *Dune 2* quatre règles fondamentales ont été identifiées : le « temps réel », le « brouillard de guerre », le « contrôle indirect » et « l'arbre des technologies ». Le « temps réel » est opposé au « tour par tour » utilisé aux échecs par exemple où, chaque joueur, l'un après l'autre, modifie l'état du jeu à sa convenance. Dans un jeu en « temps réel », chacun est libre de réaliser une action à n'importe quel instant.

Le « brouillard de guerre » cache les parties de l'environnement non encore explorées par le joueur. En conséquence, les unités de l'adversaire sont cachées tant

12. Westwood Studios, 1992

qu'elles n'entrent pas dans une zone déjà découverte par le joueur. Le principe de « brouillard de guerre » a été amélioré par la suite en prenant en compte le « champ de vision » de chaque unité. Dans ce contexte, pour être visibles, les unités de l'adversaire doivent pénétrer le champ de vision d'une unité contrôlée par le joueur et non simplement entrer dans une zone déjà explorée et non surveillée.

Le « contrôle indirect » permet au joueur de donner des ordres a une ou plusieurs de ses unités. Celles-ci vont alors tenter de les réaliser sans que le joueur ait besoin de les contrôler directement.

Enfin, « l'arbre des technologies » représente une architecture abstraite des différents éléments du jeu. En début de partie, le joueur possède un ensemble restreint d'unités et de structures qui vont lui permettre d'en construire de nouvelles et débloquer ainsi de nouveaux éléments comme des bonus de puissance, de résistance, etc.

3.1.1.3. Buts du jeu

Dans un jeu de STR, le but dépend du mode de jeu. Traditionnellement les jeux de STR en proposent deux : la **campagne** et l'**escarmouche**. Le mode campagne a pour objectif de séduire le joueur en vue de lui apprendre à jouer. Une campagne est divisée en missions qui introduisent progressivement le contenu et la complexité du jeu. Dans ce mode de jeu, le joueur devra atteindre des objectifs en lien avec le déroulement de l'histoire comme atteindre une position, résister aux attaques de l'adversaire ou construire une unité particulière.

Le mode escarmouche (non présent dans *Dune 2*) a pour objectif d'étendre la durée de vie du jeu en permettant au joueur de remporter des défis contre d'autres joueurs ou l'ordinateur. Dans ce mode de jeu, le but consiste, en général, à éliminer toutes les unités de l'adversaire.

3.1.1.4. Bilan

Bien que les principes énoncés ci-dessus nous semblent être représentatifs de la majorité des STR, certains ne les respectent pas. Cela n'affecte en rien leur appartenance à cette famille de jeu. Bien au contraire, cette diversité contribue à renouveler et à faire évoluer les jeux de STR. À titre d'exemple, il existe des STR où l'exploitation des ressources est inexistante (*Sudden Strike* [13]), où le brouillard de guerre est absent (*Ground Control* [14]) et où un contrôle direct des unités est possible (*Battlezone II* [15]). Cependant, en règle générale, dans un jeu de STR, le joueur donne des ordres à ses unités pour qu'elles réalisent différentes actions (comme se déplacer, construire, etc.).

Un jeu de STR, où ces instructions peuvent être données au travers de programmes, peut être une solution pour notre jeu sérieux. L'idée est d'inciter le joueur à réaliser des programmes informatiques qui l'assisteront durant le jeu et l'aideront à remporter la partie, s'ils sont efficaces, pertinents et adaptés au contexte du jeu.

---

13. Fireglow Games, 2000
14. Massive Entertainment, 2000
15. Pandemic Studios, 1999

3.1.2. *Les jeux de stratégie temps réel sont-ils compatibles avec l'enseignement de la programmation ?*

Comme nous l'avons vu précédemment (section 2.2.3), plusieurs approches utilisent l'activité de jeu pour inciter le joueur à apprendre la programmation. *Colobot* et les jeux de type *Robocode* proposent différentes approches pour intégrer la programmation dans un contexte de jeu :

– *Colobot* est défini par ses concepteurs comme un jeu de STR. Pourtant, les jeux temps réel sont dynamiques et requièrent une forte interaction de la part du joueur alors que l'activité de programmation nécessite du temps de conception et de réalisation. Par conséquent, les règles classiques des jeux de STR ont été modifiées dans *Colobot* pour permettre cette intégration. Le joueur ne peut contrôler directement qu'une seule entité à la fois, d'où l'utilité d'automatiser ces robots à l'aide de programmes. Le mode multijoueur est absent pour éviter les temps d'attente entre joueurs pendant les phases de programmation ;

– Les jeux de type *Robocode* distinguent l'activité de programmation de la simulation. Tout d'abord, le joueur écrit une IA pour son robot, puis l'exécute dans le contexte du jeu. Ainsi, le joueur est inactif durant la simulation et est simplement spectateur du déroulement de son programme.

Pour rendre les jeux de STR compatibles avec la pratique de la programmation, nous proposons le système Prog&Play. Ce dernier permet au joueur de concevoir des programmes capables de contrôler ses unités en vue de leur attribuer des comportements. Cette phase de conception est réalisée à la manière des jeux de type *Robocode* où le joueur écrit son IA puis la teste dans le contexte du jeu. Une fois ses programmes mis au point, le joueur peut alors les utiliser dans les différents modes de jeu (campagne et escarmouche). Tout en conservant une interaction directe sur l'environnement, le joueur peut exécuter ses programmes en cours de partie à la manière de *Colobot*.

3.1.3. *Conclusion*

Après avoir vérifié la popularité des jeux de STR auprès des étudiants de notre université, nous avons présenté ce type de jeu afin de déterminer la manière dont nous allions rendre la programmation compatible avec les jeux de STR. Nous nous inspirons des systèmes existants, tels que *Colobot* et les jeux de type *Robocode*, pour concevoir notre jeu sérieux et y intégrer nos objectifs d'apprentissage.

Les spécifications de notre jeu sérieux sont donc les suivantes : véhiculer des concepts de programmation ; être portable dans différents contextes d'enseignement ; être basé sur un jeu de STR ; conserver le *gameplay* du jeu de STR support et autoriser le contrôle des unités via des programmes informatiques.

3.2. *Implémentation*

Les jeux de STR sont des programmes extrêmement complexes, composés de plusieurs dizaines de milliers de lignes de code. Notre objectif n'étant pas de réaliser un nouveau jeu de STR, nous avons décidé d'utiliser un moteur de jeu existant. Ce moteur devait avoir son code source ouvert pour nous permettre d'y intégrer les fonctionnalités liées à notre jeu sérieux. Nous avons choisi d'utiliser le jeu *Kernel Panic* [16] lui même basé sur le moteur de jeu *Spring* [17].

3.2.1. *Caractéristique de Kernel Panic*

Kernel Panic est un jeu de STR simple avec peu de caractéristiques : il n'y a pas de gestion de ressources excepté le temps et l'espace ; toutes les unités sont gratuites ; la hiérarchie de création des unités est petite avec moins de dix unités par faction ; il utilise un rendu vectoriel original en adéquation avec l'univers du jeu. Ces caractéristiques mettent l'accent sur la stratégie et la tactique dans un jeu orienté action mais accessible à tous les joueurs. L'univers du jeu *Kernel Panic* se positionne au sein même d'un ordinateur où le joueur peut prendre le contrôle d'une des trois factions disponibles : les « Systèmes », les « Hackers » et les « Réseaux ». Chacune d'entre elles propose différentes unités comme par exemple le Bit, l'Octet et l'Assembleur chez les « Systèmes », le Virus, le Bogue et le Ver chez les « Hackers » et le Port, le Parefeur et le Paquet chez les « Réseaux ».

3.2.2. *Présentation de la campagne du jeu sérieux*

Kernel Panic est un jeu multijoueur qui ne fournit pas de mode campagne intégré. Cependant, une campagne peut s'avérer utile pour introduire progressivement les concepts de programmation non encore acquis par les étudiants.

Spring dispose d'un interpréteur Lua [18] intégré permettant aux utilisateurs de structurer des scripts au sein d'une archive chargée et exécutée dynamiquement par le moteur. Grâce à cette fonctionnalité, nous avons conçu un squelette d'archive pour permettre aux enseignants de créer des campagnes adaptées à leurs enseignements.

Nous avons donc utilisé ce squelette pour concevoir une campagne centrée sur les fondamentaux de la programmation. Cette dernière prend avantage de l'univers de *Kernel Panic* et se décompose en un ensemble de missions. Le scénario suivant est proposé aux étudiants : « *Depuis un certain nombre d'années, une guerre secrète fait rage au sein même des ordinateurs. Des attaques ont régulièrement lieu contre d'innocentes victimes. Aujourd'hui c'est votre tour. Votre agresseur a capturé le contrôleur de votre souris. Vous devez le récupérer. Votre seule solution : la programmation* ». Les cinq missions créées sont présentées en détail par Muratet *et al.* (2009). En voici, deux échantillons :

16. `http://springrts.com/wiki/Kernel_Panic` consulté le 8 Janvier 2010
17. `http://springrts.com` consulté le 8 Janvier 2010
18. `http://www.lua.org` consulté le 15 Mars 2010

– Mission 1 : « *Lors de la précédente attaque, vous avez perdu de nombreuses unités et celles qui vous restent sont dispersées sur la carte. Un seul Bit est encore sous votre contrôle. Un rapport vous est parvenu vous indiquant qu'un Octet se trouverait au point de ralliement (1056, 1792). Déplacez votre unique unité à cette position pour tenter de retrouver l'Octet perdu. Bon courage commandant…* » Pour atteindre cet objectif, le joueur doit réaliser un petit programme en manipulant des variables, des types, des affectations, des fonctions et le passage de paramètre. La figure 1 montre une solution ;

– Mission 3 : « *Toutes les unités que vous possédez ont subi de lourds dommages lors de la précédente attaque. Vous devez les réparer avant de lancer la contre-attaque. Le dernier Assembleur encore en marche et capable d'effectuer les réparations se trouve aux coordonnées (256, 1024). Déplacez toute votre armée à la rencontre de cet Assembleur* ». Dans cette mission une structure de contrôle itérative doit être utilisée pour parcourir l'ensemble des unités et les déplacer à la bonne position. La figure 2 présente une solution.

```
01 - programme mission1
02 - glossaire
03 -   Unit u
04 -   Position p
05 - début
06 -   PP_Open()
07 -   p.x <- 1056
08 -   p.y <- 1792
09 -   u <- PP_GetUnitAt(1, MY_COALITION)
10 -   PP_Unit_GiveOrder(u, MOVE, p)
11 -   PP_Close()
12 - fin
```

Figure 1 – algorithme de la mission 1.

```
01 - programme mission3
02 - glossaire
03 -   Unit u
04 -   Position p
05 -   Counter c
06 - début
07 -   PP_Open()
08 -   p.x <- 256
09 -   p.y <- 1024
10 -   pour c <- 1 jusqu'à
       PP_GetNumUnits(MY_COALITION) faire
11 -     u <- PP_GetUnitAt(c, MY_COALITION)
12 -     PP_Unit_GiveOrder(u, MOVE, p)
13 -   finPour
14 -   PP_Close()
15 - fin
```

Figure 2 – algorithme de la mission 3.

Grâce à ce jeu de missions, le joueur est acteur du déroulement de l'histoire et progresse dans le scénario après chaque objectif atteint. Nous profitons de cette dynamique pour introduire progressivement dans chaque mission un concept algorithmique ou une difficulté supplémentaire en vue de proposer de nouveaux défis en lien avec le savoir à enseigner. A titre d'exemple, la séquence est introduite lors de la première mission, la structure de contrôle conditionnelle lors de la deuxième, la structure de contrôle itérative lors de la troisième et les structures de contrôle imbriquées lors de la quatrième. La cinquième mission, plus ouverte, permet aux étudiants de concevoir et d'implémenter une stratégie de jeu à l'aide des compétences acquises lors des quatre premières missions.

Pour pouvoir réaliser ces programmes, les étudiants doivent avoir à leur disposition une API (*Applicative Programming Interface*) capable d'établir un pont entre leurs programmes et le jeu de STR de manière transparente. Pour cela, nous avons créé l'API Prog&Play dont l'interface est présentée dans le tableau 1.

PP_Open	Ouvre l'API Prog&Play et demande au jeu une première mise à jour des données
PP_Close	Ferme l'API Prog&Play
PP_Refresh	Demande au jeu de mettre à jour les données
PP_GetMapSize	Récupère la taille de la carte
PP_GetNumUnits	Récupère le nombre d'unités visibles
PP_GetUnitAt	Récupère la $i^{\text{ème}}$ unité visible
PP_Unit_Get…	Ensemble de fonctions permettant de récupérer les attributs d'une unité (position, capital santé, etc.)
PP_Unit_GiveOrder	Définit un ordre pour une unité donnée

Tableau 1 – Interface de l'API Prog&Play utilisée par le joueur.

Actuellement, il est possible de charger par l'intermédiaire de cette API les conditions de victoire, la taille de la carte, la position de départ du joueur et les unités visibles par le joueur (non cachées par le brouillard de guerre). Pour chaque unité, le joueur peut récupérer sa coalition, son type, sa position, sa santé, son groupe et les actions en cours de traitement. Le choix des données transférées à travers l'API Prog&Play et les fonctions d'interface proposées sont des points critiques toujours en cours d'évolution. L'API est pour l'instant disponible dans les langages C, C++, Java, Ada, OCaml, Scratch et un langage interprété nommé Compalgo (utilisé dans des enseignements spécifiques de notre université). Ainsi le jeu sérieux est utilisable pour différentes approches pédagogiques comme l'approche impérative, orienté objet et fonctionnelle.

3.2.3. *Architecture fonctionnelle du jeu sérieux*

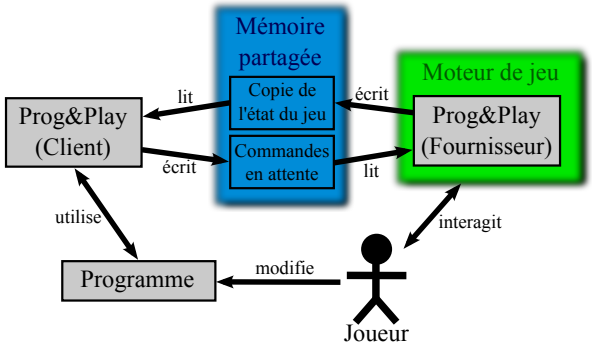


Figure 3 – Architecture fonctionnelle du jeu sérieux.

D'un point de vue technique, la communication entre le programme du joueur et le jeu vidéo est réalisée par l'intermédiaire d'une mémoire partagée. La figure 3 montre l'architecture fonctionnelle du jeu sérieux où l'API Prog&Play sert d'interface à la gestion de cette mémoire partagée pour simplifier la communication et la

synchronisation entre le programme du joueur et le moteur de jeu. Deux interfaces sont donc disponibles : la partie « Client » est utilisée par le joueur pour développer son propre programme et la partie « Fournisseur » est intégrée au moteur du jeu. Sur demande du programme du joueur, les données pertinentes du jeu sont copiées dans la mémoire partagée. Pour éviter des situations incohérentes, le programme du joueur travaille sur cette copie pour lire les données et écrire les commandes à travers l'interface « Client ». La mémoire partagée est régulièrement vérifiée par le moteur de jeu pour réaliser les actions en attente. De cette manière, à n'importe quel instant, le joueur peut stopper l'exécution de son programme, le modifier et le relancer pour qu'il se reconnecte à la mémoire partagée sans perturber le déroulement du jeu.

Le « Fournisseur » a donc en charge la gestion de la mémoire partagée. Dans un premier temps, le « Fournisseur » créé et initialise la mémoire partagée de manière à ce qu'elle soit accessible par le « Client ». Après cette initialisation, le jeu entre dans la boucle de simulation et à chaque itération le « Fournisseur » consulte la mémoire partagée pour savoir si un rafraîchissement est demandé. Si tel est le cas, le « Fournisseur » calcule la mise à jour en fonction de l'état courant du jeu et la copie dans la mémoire partagée. Le « Fournisseur » s'attache ensuite à traiter les commandes définies par le « Client ». Enfin lorsque le jeu est terminé, le « Fournisseur » nettoie et libère la mémoire partagée.

Du côté « Client », l'appel aux fonctions de l'API peut être réalisé après s'être connecté à la mémoire partagée et avoir demandé un premier rafraîchissement. Dans le cas contraire, les fonctions de l'API retournent un code d'erreur ou lèvent une exception en fonction du langage utilisé. Lors d'une demande de rafraîchissement, cet appel est bloquant tant que le « Fournisseur » n'a pas détecté et réalisé la mise à jour.

4. Évaluation

Le jeu sérieux étant opérationnel, il reste à évaluer son efficacité et sa pertinence en contexte d'enseignement. Pour ce faire, nous définissons, dans la sous-section 4.1, la méthodologie de l'ingénierie didactique qui nous sert de cadre théorique. Puis, nous présentons dans la sous-section 4.2 une mise en œuvre itérative de cette méthodologie. Finalement, nous définissons les critères d'évaluation et nous donnons les résultats des expérimentations dans la sous-section 4.3.

4.1. *Cadre théorique*

Nous proposons d'étudier l'intérêt d'un jeu sérieux pour enseigner la programmation et attirer les étudiants vers l'informatique. Pour atteindre cet objectif, nous proposons d'utiliser la méthodologie de l'ingénierie didactique (Cobb *et al.*, 2003) : « *prototypically, design experiments entail both "engineering" particular forms of learning and systematically studying those forms of learning within the context defined by the means of supporting them. This designed context is subject to test and*

revision, and the successive iterations that result play a role similar to that of systematic variation in experiment ». Cette méthodologie vise à expérimenter de nouvelles formes d'apprentissages à travers la mise en œuvre de moyens spécifiques. Dans cette approche, le chercheur tente de valider une théorie « humble » concernant les processus d'apprentissages et envisage d'effectuer une série d'expérimentations pour la tester. Pour chaque itération, le principe consiste à construire, en collaboration avec les protagonistes, une ingénierie didactique, à la mettre en œuvre au cours d'une ou plusieurs expérimentations en procédant à de multiples observations puis à analyser les résultats afin de proposer une éventuelle nouvelle ingénierie didactique destinée à tester de nouveaux éléments concernant la théorie. Chaque itération permet de faire varier différents paramètres supportés par la théorie. L'objectif n'est pas de construire une ingénierie didactique idéale mais bien de valider une théorie concernant l'apprentissage.

Dans notre cas, il s'agit d'une part de valider l'hypothèse selon laquelle les jeux sérieux peuvent susciter l'intérêt des étudiants envers la programmation et d'autre part de déterminer les contextes d'enseignements pour lesquels le jeu sérieux créé donne des résultats positifs.

4.2. *Évaluation itérative*

L'ingénierie didactique est basée sur un processus itératif qui permet au jeu sérieux et aux évaluations associées d'évoluer. Chaque itération est composée de trois étapes : concevoir l'ingénierie didactique, la mettre en œuvre et conduire une analyse rétrospective. Lors de la première étape, l'objectif de l'itération doit être clarifié. Suite à cette première étape, un enseignement doit être trouvé pour conduire l'expérimentation en accord avec les objectifs fixés pour cette itération. Puis, en fonction du contexte d'enseignement, une ingénierie didactique appropriée doit être formulée par l'équipe de recherche, les enseignants ou les deux. Finalement, les critères d'évaluation et les indicateurs associés sont définis. La deuxième étape consiste à réaliser l'expérimentation et à collecter un maximum de données afin de garantir une analyse rétrospective rigoureuse. Finalement, la troisième étape porte sur l'analyse des données afin de vérifier les objectifs fixés lors de cette itération et proposer éventuellement de nouvelles expérimentations. Trois itérations ont pour l'instant été réalisées avec ce jeu sérieux.

4.2.1. *Première itération*

L'objectif de cette première itération est d'observer le comportement des étudiants vis-à-vis du jeu sérieux afin de déterminer son potentiel motivationnel et son influence sur les activités des enseignants. Pour réaliser cette première itération, nous avons choisi de tester le jeu sérieux dans un environnement familier sous la forme d'un atelier de programmation. Cette expérimentation s'est déroulée avec des étudiants en première année de l'Institut Universitaire de Technologie (IUT) A de Toulouse. Sur une promotion de 196 étudiants, 40 se sont portés volontaires pour participer à cet atelier.

Parmi eux 15 étudiants représentatifs de la population de l'IUT ont été sélectionnés [19]. Les étudiants étaient novices en programmation : au moment de l'expérimentation, ils ne connaissaient aucun langage de programmation à l'exception du « Compalgo », un langage algorithmique développé par les enseignants de l'IUT A. Durant 5 séances d'une heure et trente minutes, les étudiants ont utilisé le jeu avec Compalgo dans un environnement Windows en présence d'un enseignant.

Pour cette expérimentation, les enseignements ont été structurés en deux phases. Durant la première, les étudiants jouent au jeu en mode multijoueur sans utiliser la programmation. Ainsi, ils se familiarisent avec l'univers et les unités du jeu. La seconde phase est une présentation de l'API Prog&Play que les étudiants apprennent à utiliser à travers la résolution des missions. Après ces deux phases, les étudiants devraient être capables de réaliser des programmes simples compatibles avec *Kernel Panic*.

4.2.2. *Deuxième itération*

L'objectif de la deuxième expérimentation est d'étudier la mise en œuvre du jeu sérieux à une plus grande échelle dans le but d'évaluer son influence sur les résultats des étudiants et de récolter l'appréciation des enseignants. En effet, les enseignants prenant part à cette expérimentation sont externes à ce projet de recherche et n'ont aucune expérience dans les jeux sérieux. Pour réaliser cette deuxième itération, nous introduisons le jeu sérieux au premier semestre de la licence informatique de l'Université Paul Sabatier (Toulouse). Dans cet enseignement, la programmation est abordée par une approche fonctionnelle. Les étudiants étudient les bases de ce paradigme de programmation durant six séances de travaux pratiques (TP) de deux heures chacun. Le langage « OCaml » est utilisé dans un environnement Linux.

La promotion est composée de plus de 300 étudiants [20], structurée en groupes de TP de 15 étudiants. La distribution des étudiants dans ces groupes de TP est réalisée de manière aléatoire. Environ 20 enseignants participent à l'encadrement des TP. Pour évaluer l'influence du jeu sérieux sur les résultats des étudiants, la promotion est divisée en deux groupes de 150 étudiants. Le premier groupe, dit groupe témoin, suit l'enseignement classique alors que le second groupe, dit groupe Prog&Play, suit l'enseignement dans lequel le jeu sérieux est intégré.

Pour valider l'intégration du jeu dans la formation, les deux groupes d'étudiants doivent aborder le même contenu. Par conséquent, le jeu sérieux doit être en mesure de proposer le même type d'exercice que dans la formation classique. Sans ces conditions, l'équipe pédagogique se réserve le droit de ne pas intégrer le jeu dans les enseignements. Le tableau 2 montre les objectifs d'apprentissages pour les TP classiques.

---

19. 3 filles et 12 garçons ; 19 ans de moyenne d'âge ; 10 baccalauréats scientifiques, 4 baccalauréats techniques et 1 non communiqué.

20. 19 ans de moyenne d'âge ; 80% de garçons ; baccalauréats scientifiques 85%, économique et social 2%, littéraire 1%, technique 7%, professionnel 2%, autre 3%.

Tableau 2 – Objectifs d'apprentissage pour les TP.

TP	Objectifs
1	Présentation de l'environnement de travail : système d'exploitation (Linux - Mandriva 2008), gestionnaire de fenêtres (KDE), la boucle interactive d'OCaml et le traitement de texte Kate.
2	Fonctions arithmétiques : le filtrage, les n-uplets, la fermeture et la récursivité ; introduction à la dichotomie.
3	Utilisation de la bibliothèque graphique : définition d'une bibliothèque, trigonométrie, optimisation et récursivité.
4	Manipulation de listes : gestion et construction de listes ; filtrage et récursivité sur les listes ; introduction aux prédicats ; tris.
5 et 6	Manipulation de structures de données : ré-utilisation des fonctions précédentes ; projection, suppression de doublons, sélection, mise à jour et tri ; manipulation de type utilisateur.

Pour le groupe Prog&Play, les séances de TP 1, 3 et 4 ont été revus. Après validation par l'équipe pédagogique, les modifications apportées sont présentées ci-dessous :

– **TP 1** : Outre la présentation de l'environnement de travail, le premier TP est utilisé pour présenter le jeu et son univers. Il explique d'une manière détaillée la procédure à suivre pour réaliser des parties en mode multijoueur. Les étudiants sont invités à tester le jeu en séances libres avant la troisième séance de TP.

– **TP 3** : La troisième séance de TP est centrée sur l'utilisation d'une bibliothèque. Le support a donc été totalement réécrit pour présenter l'API Prog&Play ainsi que la campagne. Afin de respecter le type d'exercice présenté dans le support originel, une nouvelle mission (notée 1a) a été ajoutée à la campagne entre la première et la deuxième mission. Ainsi, à la fin de la première mission, l'Octet perdu ne se trouve pas à la position prévue. De nouvelles informations sont alors fournies au joueur pour rechercher l'Octet. Dans cette nouvelle mission, le joueur contrôle donc seulement un Bit (le même que pour la mission 1) et connaît dans quelle direction et à quelle distance se trouve l'Octet par rapport au Bit. Le joueur doit alors calculer la position de l'Octet en fonction de la position du Bit et des informations fournies. Lors de cette troisième séance, les étudiants sont invités à réaliser les quatre premières missions de cette nouvelle campagne. Par cet intermédiaire, les étudiants s'initient à l'utilisation d'une bibliothèque avec la mission 1, réalisent un exercice de trigonométrie dans la mission 1a, manipulent le filtrage dans la mission 2 et la récursivité dans la mission 3.

– **TP 4** : Enfin, la quatrième séance de TP a également été entièrement réécrite. Lors de cette séance les étudiants travaillent à la réalisation de la mission 4. Ils commencent par créer des fonctions pour manipuler des listes d'unités et les utilisent pour terminer une première fois la mission. À l'aide d'algorithmes de tris, les étudiants réalisent une seconde solution plus efficace et entrevoient ainsi le principe de l'optimisation, non traité lors du TP 3 modifié. La dernière mission de la campagne est à réaliser en séance libre.

En raison du nombre important d'enseignants prenant part à cette expérimentation, une séance préalable leur a été proposée pour leur présenter le jeu, les modifications apportées aux sujets de TP et la solution des exercices.

4.2.3. *Troisième itération*

Cette itération a pour objectif d'étudier l'utilisabilité du jeu par de tierces personnes. A cette occasion, nous avons conçu un site Web où les enseignants peuvent accéder au jeu sérieux et l'adapter à leur pédagogie. Deux expérimentations ont pu être conduites dans ce contexte.

La première expérimentation a été réalisée en collaboration avec un enseignant du département Génie Électrique et Informatique Industrielle (GEII) de l'IUT A (Toulouse). Cet enseignant a choisi d'utiliser le jeu avec des étudiants en difficulté et attend du jeu qu'il puisse motiver les étudiants à programmer. Quinze étudiants ont utilisé le langage C++ pour réaliser la campagne en une dizaine d'heures sous l'encadrement de l'enseignant.

La seconde expérimentation a été réalisée en collaboration avec deux enseignantes de l'Université Pierre et Marie Curie (UPMC - Paris VI). Elles ont déployé le jeu dans une unité d'enseignement orientée TICE (Technologies de l'Information et de la Communication pour l'Education). Dans cette unité d'enseignement, les enseignants proposent un ensemble de projets dont un est basé sur notre jeu sérieux. Parmi les volontaires, huit étudiants qui avaient suivi le cours d'introduction à la programmation impérative ont été retenus. Les enseignantes ont supervisé deux séances de deux heures pour introduire le jeu et les concepts de programmation non étudiés en cours (utilisation d'une librairie et des enregistrements - struct du langage C). Suite à cette présentation, les trois premières missions sont traitées à l'aide d'un langage algorithmique puis codés en langage C. Enfin, les étudiants avaient deux mois pour terminer la campagne en autonomie et préparer une présentation de leurs réalisations.

4.3. *Critères d'évaluation et résultats*

Les trois itérations présentées ci-dessus ont été conçues pour satisfaire les objectifs suivants : déterminer le potentiel motivationnel du jeu sérieux, évaluer son influence sur les résultats des étudiants et rapporter le point de vue des enseignants.

À partir de ces objectifs, nous définissons quatre critères d'évaluation : la performance des étudiants ; l'utilisabilité du système ; l'amusement ; et l'appréciation des enseignants. Le premier et le troisième critère évaluent le concept de « jeu sérieux » sous l'angle de l'apprentissage de la programmation et du divertissement procuré par le jeu. Le deuxième critère, permet d'identifier les composants du jeu qui peuvent perturber les étudiants (problèmes d'installation, bogues, etc.). Le dernier critère permet d'obtenir une évaluation qualitative du jeu sérieux par les enseignants.

Avant de présenter chaque critère et les résultats associés, nous synthétisons dans la figure 4 tous les moyens mis en œuvre pour collecter les données. Cette opération

a été organisée en trois étapes : avant, pendant et après l'utilisation du jeu sérieux. À noter que tous ces moyens n'ont pas été utilisés à chaque itération.

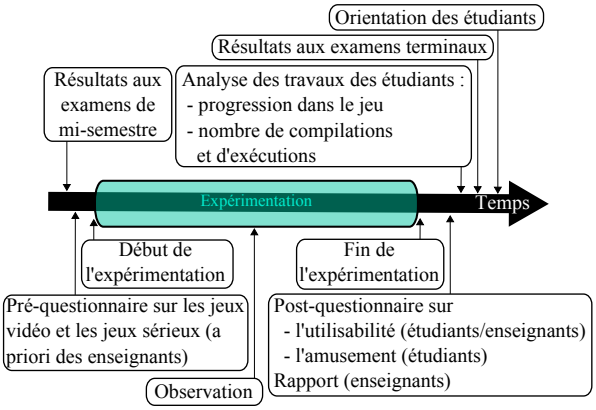


Figure 4 – Synthèse des moyens utilisés pour collecter les données durant les expérimentations.

4.3.1. *Critère 1 : performance des étudiants*

L'évaluation de l'apprentissage est basée sur les travaux de Chen *et al.* (2007), Gestwicki *et al.* (2008) et Leutenegger *et al.* (2007). Nous définissons trois indicateurs : la quantité de travail réalisée, les connaissances acquises et l'orientation des étudiants. La quantité de travail réalisé est évaluée en utilisant la progression du jeu (nombre de missions terminées). Les connaissances acquises sont évaluées avec l'évolution des résultats des étudiants entre l'examen de mi-semestre et l'examen terminal.

Lors de la première itération, nous avons analysé le premier indicateur (quantité de travail réalisée) en traitant les ressources produites par les étudiants pendant l'expérimentation. Nous avons comptabilisé le nombre de missions terminées par chaque étudiant et pour chaque mission, le nombre de compilations et d'exécutions nécessaires à la réalisation des solutions. Ces données ont pu être récupérées après avoir intégré un « espion » à l'environnement de développement utilisé lors des séances d'enseignement. Grâce à ces données, nous pouvons définir, pour chaque mission « m », un indice de difficulté (noté « d_m ») présenté dans la figure 5. Cet indice de difficulté est calculé comme indiqué dans l'équation 1, où « $nbEtudiants_m$ » est le nombre d'étudiants ayant terminé la mission « m » et « $nbCompilations_{im}$ » et « $nbExecutions_{im}$ » sont respectivement le nombre de compilations et d'exécutions requises par le i$^{\text{ème}}$ étudiant pour compléter la mission « m ».

$$d_m = \frac{\sum_{i=1}^{nbEtudiants_m} nbCompilations_{im} + nbExecutions_{im}}{nbEtudiants_m} \quad [1]$$

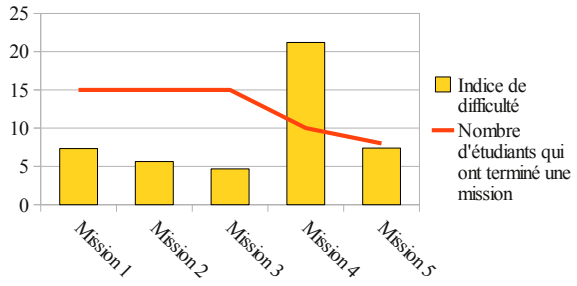


Figure 5 – Indice de difficulté pour chaque mission.

Contrairement aux attentes, l'accroissement de la difficulté des missions n'est pas régulier. Un écart important existe entre la troisième et la quatrième mission. Dans l'avenir, nous pensons diviser la mission 4 en deux nouvelles missions afin de proposer un défi plus progressif. La baisse de l'indice de difficulté entre les missions 1 et 3 montre l'appropriation progressive de l'API par les étudiants et leur capacité à la combiner avec des structures de contrôle simples. Nous envisageons également de mettre à jour la dernière mission afin de proposer un défi plus intéressant pour terminer la campagne. En effet, les étudiants qui ont atteint cette mission ont été frustrés par sa simplicité. Johnson *et al.* (2005) argumentent sur l'importance de l'histoire dans un jeu vidéo qui maintient l'intérêt du joueur, l'encourage à progresser constamment et lie ces actions aux objectifs futurs. Les défis sont introduits par l'intermédiaire d'une histoire et doivent correspondre à l'expérience des joueurs. Greitzer *et al.* (2007) mettent en évidence la difficulté de construire un scénario ni trop facile ni trop difficile afin de proposer un défi intéressant sans décourager le joueur.

Les autres indicateurs (connaissances acquises et orientation des étudiants) ont été évalués durant l'expérimentation de la deuxième itération. Concernant les « connaissances acquises », nous avons récupéré les résultats des étudiants aux examens de mi-semestre et terminaux. Ces examens ont été conçus par l'équipe pédagogique externe au projet et sont les mêmes pour tous les étudiants. La figure 6 montre la note moyenne pour le groupe témoin et le groupe Prog&Play. Nous précisons que les résultats à l'examen de mi-semestre n'étaient pas connus au début de l'expérimentation.

L'analyse des données récoltées montre une baisse des résultats entre l'examen de mi-semestre et l'examen terminal quelque soit le groupe considéré. Cependant, cette baisse est moins importante pour le groupe Prog&Play (1,25 points) que pour le groupe témoin (1,55 points). Nous ne pouvons attribuer ce résultat seulement à l'utilisation du jeu sérieux car les expérimentations réalisées en contexte réel comportent de nombreux paramètres incontrôlables. Néanmoins, nous considérons que le jeu sérieux a contribué à ce résultat positif.

Concernant « l'orientation des étudiants » au second semestre, neuf majeures leurs sont proposées dont une est dédiée à l'enseignement de l'informatique. Parmi les étu-

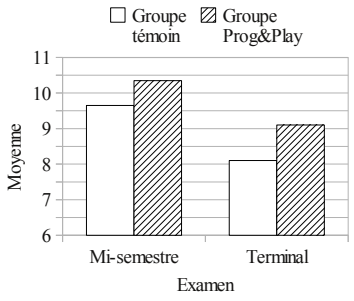


Figure 6 – Notes moyennes pour chaque examen en fonction du groupe (20 est la note maximale).

diants du groupe Prog&Play, 55% ont choisi cette majeure contre 49% pour le groupe témoin.

Ainsi, ces résultats montrent d'une part que la campagne offre une complexité suffisante pour encourager le travail et stimuler l'intérêt des étudiants, même si la difficulté des missions est moins progressive que prévue. D'autre part, les données collectées révèlent que le jeu sérieux a contribué à réduire l'échec des étudiants comparé au groupe témoin et a permis de recruter plus d'étudiants pour la majeure informatique au second semestre.

4.3.2. *Critère 2 : utilisabilité du système*

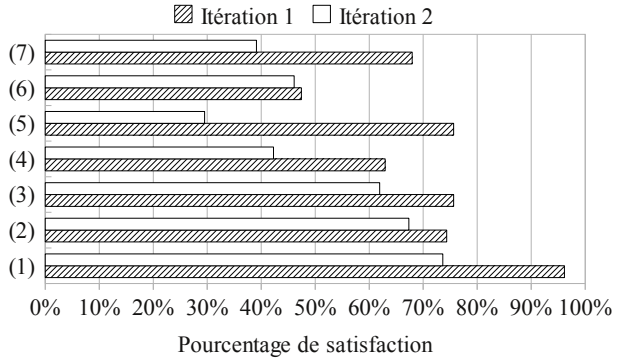
L'évaluation de l'utilisabilité du système est basée sur un questionnaire post-expérimental dans lequel nous demandons aux étudiants d'évaluer le jeu sérieux. Grâce aux travaux de Siang *et al.* (2003), nous avons élaboré un questionnaire pour évaluer la hiérarchie des besoins du joueur afin d'identifier les parties du jeu sérieux nécessitant des améliorations. Cette hiérarchie est divisée en sept niveaux où les premiers servent de base aux niveaux supérieurs. Les sept niveaux par ordre de priorité sont les suivants : (1) le **besoin de règles**, les joueurs recherchent des informations pour comprendre les règles de base structurant le jeu ; (2) le **besoin de sûreté**, les joueurs ont besoin de trouver de l'aide sur le fonctionnement du jeu ; (3) le **besoin d'appartenance**, les joueurs ont besoin de s'approprier le jeu pour se sentir capable d'atteindre les objectifs ; (4) le **besoin d'estime**, les joueurs ont besoin d'être valorisés par le jeu (*feedback*, progression, score, compétition, etc.) ; (5) le **besoin de connaître et de comprendre**, les joueurs ont besoin de découvrir les informations/bonus cachés et de les mettre en relation en vue de les réinvestir en situation de jeu ; (6) le **besoin d'esthétique**, les joueurs ont besoin de beaux graphismes, d'effets visuels, d'une musique appropriée, d'effets sonores, etc. ; (7) le **besoin d'auto accomplissement**, les joueurs veulent être capables de projeter leur créativité et imagination dans le jeu sous contrainte du respect des règles.

Outre ce premier questionnaire, nous demandons aux étudiants d'exprimer leurs critiques, remarques et suggestions sur le jeu sérieux. Nous les questionnons également sur la démarche employée en leur permettant de donner leur point de vue sur la pertinence à utiliser un jeu sérieux pour l'apprentissage de la programmation. Ces questions sont présentées dans la figure 7.

Q1 : Selon vous, y-a-t'il un intérêt à utiliser un jeu vidéo pour apprendre à programmer ? (1=« Pas du tout », 7=« Tout à fait »)	Q2 : Pensez-vous que des TP basés sur un jeu vidéo ont leur place dans cette formation ? (1=« Pas du tout », 7=« Tout à fait »)

Figure 7 – Perception des étudiants de l'utilité d'un jeu vidéo pour apprendre la programmation.

Lors des deux premières itérations, nous avons évalué la hiérarchie des besoins du joueur. La figure 8 montre le niveau de satisfaction des étudiants pour chaque niveau de la hiérarchie.



(1) besoin de règles, (2) besoin de sûreté, (3) besoin d'appartenance
(4) besoin d'estime, (5) besoin de connaître et de comprendre
(6) besoin d'esthétique, (7) besoin d'auto accomplissement

Figure 8 – Satisfaction des étudiants à propos de la hiérarchie des besoins du joueur.

Concernant la première itération, le niveau de satisfaction le plus faible porte sur le besoin d'esthétique avec une valeur moyenne de 47%. Le graphisme décalé de *Kernel Panic* est moyennement apprécié des étudiants. Néanmoins, ce STR simplifié permet une prise en main rapide qui influence de manière positive la satisfaction des premiers niveaux de la hiérarchie.

Pour la deuxième itération, nous observons des niveaux de satisfaction inférieurs à la première itération alors que le jeu sérieux était identique à l'exception de la nouvelle mission (1a). Afin d'expliquer cette différence, nous avons analysé les remarques et critiques exprimées par les étudiants. Les principales critiques portaient sur la lourdeur

des sujets de TP quelque soit le groupe considéré. Pour le groupe Prog&Play ce sentiment était d'autant plus frustrant que cette lourdeur empiétait sur le jeu. Contrairement à la première itération, les étudiants ont donc eu plus de difficultés à comprendre les bases du jeu et les objectifs de chaque mission car les problèmes posés étaient amenés par les sujets de TP et non pas par le jeu lui même. Ainsi, cette expérimentation met en évidence que les résultats obtenus dépendent non seulement du jeu sérieux lui même mais également de la mise en œuvre, de l'encadrement, des supports de cours, etc.

Néanmoins, quelque soit l'itération, les étudiants ont apprécié l'initiative et sont demandeurs de ce type de pédagogie. Les réponses moyennes aux questions de la figure 7 sont présentées dans le tableau 3 pour les deux premières itérations.

Tableau 3 – Réponses moyennes aux questions présentées figure 7 pour les deux premières itérations (valeurs comprises entre 1 et 7).

	1ère itération		**2ème itération**	
	Q1	**Q2**	**Q1**	**Q2**
Moyenne	6.17	6	5.05	5.18

Le jeu est donc fonctionnel car aucun bogue critique n'a été révélé lors des expérimentations. Seuls quelques enseignants ont rencontré des difficultés pour configurer le jeu avec leur système d'exploitation.

4.3.3. *Critère 3 : amusement*

Ce critère a été évalué grâce à un questionnaire supplémentaire distribué aux étudiants à la fin de chaque expérimentation. Les questions sont présentées dans la figure 9. La quatrième question (Q4) est notamment cruciale pour nous. Introduire des concepts algorithmiques dans le jeu ne doit pas en réduire le divertissement. Cette condition est due à la définition des jeux sérieux qui doivent être avant tout divertissants.

Q3 : Avez-vous apprécié le scénario de la campagne (les missions) ? (1=« Pas du tout », 7=« Beaucoup »)
Q4 : Pensez-vous que l'utilisation de la programmation dans *Kernel Panic* augmente le divertissement ?
☐ Réduit le divertissement ☐ Ne change rien
☐ Augmente le divertissement ☐ Je ne sais pas

Figure 9 – Questions relatives à l'amusement.

Concernant la première itération, la réponse moyenne à la troisième question (Q3, figure 9) est 5,85. Ces résultats montrent que les étudiants ont apprécié la campagne. D'une manière informelle, nous pensons que l'intérêt porté aux missions est en partie dû à la première phase de l'expérimentation. En effet, la session de jeu en réseau a permis aux étudiants de s'approprier l'univers du jeu et a rendu l'immersion des étudiants dans le scénario plus facile. Pour la quatrième question (Q4, figure 9) 100%

des étudiants ont trouvé que l'utilisation de la programmation dans le jeu augmentait le divertissement.

Concernant la seconde itération, les résultats sont légèrement inférieurs. La réponse moyenne à la troisième question (Q3, figure 9) est 4,14. Pour la quatrième question (Q4, figure 9), 42% des étudiants de cette enquête votent pour une augmentation, 22% n'indiquent aucun changement, 21% ne se prononcent pas et 15% trouvent que cela le réduit. Par conséquent, 64% des étudiants estiment que le jeu sérieux est au moins aussi divertissant que le jeu sans la programmation. Nous attribuons la baisse de ces résultats aux mêmes raisons identifiées lors de l'analyse du deuxième critère : les supports de TP perçus comme trop descriptifs et trop dirigistes ont contrebalancé l'amusement.

4.3.4. *Critère 4 : appréciation des enseignants*

Afin d'observer les activités des enseignants lors des expérimentations, nous proposons de filmer les séances. Une attention particulière doit être portée sur les activités liées à l'utilisation du jeu et à l'enseignement. L'activité nommée « utilisation du jeu » concerne des explications sur le contrôle du jeu (comment lancer le jeu ? Comment déplacer la caméra ?) et sur l'interaction entre les programmes des étudiants et le jeu (comment accéder ou commander une unité ? Comment vérifier si le programme est correct ?). L'activité « contenu d'apprentissages » concerne les explications sur les obstacles de programmation tels que les variables (type, affectation, enregistrement), les fonctions (appel, passage de paramètres), ou les structures de contrôle (conditionnelles, itératives). L'activité « autres tâches » concerne les tâches administratives (accueil des étudiants et appel), des explications sur l'utilisation du système d'exploitation, etc.

Pour compléter ces observations, nous proposons un questionnaire pré-expérimental à destination des enseignants pour recueillir leurs a priori sur les jeux vidéo et les jeux sérieux. Enfin, chaque enseignant est invité à nous faire parvenir un retour d'expérience sous la forme d'un rapport. À cette occasion, nous leur demandons de présenter les difficultés rencontrées lors de l'utilisation du jeu sérieux et d'indiquer leur point de vue quant à une éventuelle reconduite de l'expérimentation l'année suivante.

L'observation des activités de l'enseignant a été réalisée au cours de la deuxième séance de TP lors de la première itération. Le temps passé par l'enseignant pour chaque activité est détaillé dans le tableau 4. Certaines interventions de l'enseignant sont comptabilisées dans différentes activités, ce qui explique pourquoi la somme des durées de chaque activité est supérieure à la durée d'une séance (1h 30m).

Au regard des activités énoncées précédemment, le cœur de l'enseignement repose sur le contenu de l'apprentissage. Cette répartition est opportune, car, les explications liées au jeu ne doivent pas empiéter de manière excessive sur les activités de l'enseignant. D'autre part, le temps consacré à l'utilisabilité du jeu diminue au fur et à mesure que les étudiants deviennent experts dans la manipulation du jeu. Néanmoins, cette activité est importante principalement pour les premières séances. En effet, le

Activité	Durée
Utilisabilité du jeu	22m 13s
Contenu d'apprentissages	1h 3m 42s
Autres tâches	41m 27s

Tableau 4 – Durée des activités pour une séance d'enseignement.

deuxième niveau de la hiérarchie des besoins du joueur (besoin de sûreté) est principalement satisfait par l'aide des enseignants. Actuellement le jeu sérieux doit être utilisé comme un outil pour pratiquer la programmation ; les informations sur le contenu d'apprentissage ou l'utilisabilité du jeu ne sont pas incluses dans le jeu et doivent être apportées par l'enseignant ou le support de cours.

Concernant la deuxième itération, nous avons pris avantage du nombre important d'enseignants pour leur demander leurs a priori sur les jeux vidéo et les jeux sérieux avant le début de l'expérimentation. Trois femmes et douze hommes ont répondu à nos questions. Dans cet échantillon, neuf personnes attestent avoir une expérience de joueur. Concernant les jeux vidéo, les enseignants ont principalement relevé trois qualités : la capacité des jeux vidéo à favoriser la réflexion à travers le développement de stratégies de jeu, à détendre et à permettre l'immersion dans des environnements virtuels. Dans une moindre mesure, l'aspect compétition à travers le mode de jeu multijoueur, le scénario, la motivation et la créativité sont notés comme étant des qualités intrinsèques aux jeux vidéo. Concernant les défauts, la principale inquiétude porte sur le risque à l'addiction et à la coupure de la réalité. Certains enseignants les trouvent complexes et les associent à une perte de temps. Concernant les jeux sérieux, six enseignants ne se prononcent pas, sept portent un jugement positif et deux négatif. Les a priori positifs positionnent le jeu sérieux comme un outil concret, proche du centre d'intérêt des étudiants qui permet une visualisation immédiate de leurs programmes dans le contexte du jeu. Mais la principale hypothèse positive porte sur la capacité du jeu sérieux à augmenter la motivation des étudiants à travailler en informatique. Les quelques a priori négatifs se fondent sur la crainte : de donner une image erronée de l'informatique qui ne se cantonne pas aux jeux vidéo ; de favoriser principalement les garçons ; de perdre du temps à expliquer le jeu au détriment des concepts algorithmiques ; et de détourner les étudiants du fond en raison de la dimension ludique du jeu. Les enseignants sans opinion sont en attente de résultats.

Toujours par rapport à la deuxième itération, les rapports des enseignants sont mitigés. Du point de vue des performances, quelques enseignants pensent que les séances de TP basées sur le jeu sérieux ont été contre-productives pour de nombreux étudiants (il faut noter que ce ressenti ne se retrouve pas dans les résultats aux évaluations présentées figure 6). Ils n'attribuent pas cette conséquence directement au jeu sérieux mais plutôt aux supports de TP. A contrario, la plupart des enseignants ont noté une influence positive du jeu sur la motivation des étudiants.

Concernant la troisième itération, les enseignants qui ont réalisé les expérimentations par eux-mêmes donnent un rapport plus positif. Premièrement, l'enseignant du département GEII indique que la campagne du jeu sérieux est bien adaptée au profil de ses étudiants. Selon lui, les étudiants ont montré leur implication en travaillant chez eux alors que cela n'était pas demandé. Deuxièmement, les enseignantes de l'UPMC ont d'abord été impressionnées par la diversité des stratégies envisagées et des exposés préparés. Elles rapportent que les étudiants ont apprécié le projet car ils ont eu le sentiment de programmer quelque chose de concret. Les enseignantes ajoutent tout de même que la préparation des séances leur a demandé beaucoup de temps en raison de leur inexpérience dans le jeu vidéo. Dans tous les cas, ces enseignants envisagent de réutiliser le jeu pour les années à venir.

5. Conclusion

Le jeu sérieux présenté dans cet article est basé sur un jeu de STR. Par souci d'efficacité, nous avons décidé d'utiliser le moteur *Spring* associé au jeu *Kernel Panic*. Nous avons ensuite modifié le moteur de jeu pour permettre une programmation sécurisée et interactive à travers l'API Prog&Play. Cette dernière est compatible avec plusieurs langages de programmation.

Sur cette base, la mise en œuvre de notre jeu sérieux a consisté à concevoir une campagne, divisée en missions, pour intégrer progressivement les concepts liés aux fondamentaux de la programmation.

À travers un cadre théorique, nous avons conçu une évaluation itérative basée sur un ensemble d'expérimentations. Lors de la première itération, nous avons identifié que le jeu sérieux est apprécié des étudiants. La question de la portabilité du jeu s'est alors posée. Ainsi, la seconde itération a été conçue pour étudier la mise en œuvre du jeu à une échelle supérieure. Malgré des supports de TP perfectibles, le jeu sérieux a contribué à réduire l'échec des étudiants par rapport au groupe témoin. Enfin, la dernière itération avait pour objectif de laisser des enseignants intégrer le jeu par eux-mêmes dans leurs enseignements. Les rapports de ces enseignants sont positifs puisqu'ils envisagent de reconduire les expérimentations.

Le jeu sérieux est donc fonctionnel et motivant pour les étudiants. Nous attribuons ce succès à l'approche originale de notre jeu qui consiste à utiliser des programmes informatiques pour interagir avec un STR complexe. Le scénario de la campagne a également contribué à ce résultat de deux façons. Premièrement, il motive les étudiants en les rendant acteurs du déroulement de l'histoire. Deuxièmement, il introduit progressivement le contenu pédagogique. À la fin de la campagne, les étudiants maîtrisent l'API Prog&Play et sont capables de réaliser leurs propres stratégies.

Toutefois, il convient de ne pas généraliser ces résultats, en effet, il n'existe pas à l'heure actuelle de méthode pédagogique efficace pour tous les étudiants quel que soit le contexte d'apprentissage. Le jeu sérieux est une solution parmi tant d'autres et l'expérience accumulée au cours des expérimentations permet de définir un cadre dans

lequel l'utilisation de ce jeu sérieux est envisageable. Tout d'abord, le jeu sérieux semble trouver sa place en tant que complément aux formations classiques. Le soutien ou des ateliers de programmation sont particulièrement bien adaptés à son déploiement, car ils laissent le temps aux étudiants de concevoir et réaliser leurs solutions aux problèmes posés. Enfin, intégrer le jeu à une formation suppose une double adaptation : premièrement le jeu doit être ajusté au profil des étudiants via l'interface de programmation et deuxièmement le calendrier des enseignements de la formation devra être modifié pour laisser une place suffisante au jeu. Nous pensons que le respect de ces trois recommandations favorisera une mise en œuvre réussie du jeu sérieux comme élément de dynamisation des formations informatiques, de motivation et de facteur de réussite pour les étudiants.

Le système Prog&Play, *Kernel Panic* et les interfaces de programmation compatibles avec le jeu sont disponibles à l'adresse suivante : `http://www.irit.fr/~Mathieu.Muratet/progAndPlay.php`. N'hésitez pas à nous contacter si vous souhaitez obtenir de plus amples informations.

Remerciements

Ce travail n'aurait pas été possible sans la collaboration des différentes équipes pédagogiques. Les auteurs tiennent à remercier les personnes suivantes ainsi que leurs institutions de rattachement : Christian Percebois et Max Chevalier (département Informatique) et Jérémie Guiochet et André Lozes (département Génie Électrique et Informatique Industrielle) de l'IUT A de Toulouse ; Mathias Paulin, Véronique Gaildrat et tous les enseignants de l'Université Paul Sabatier qui ont participé aux expérimentations ; Elisabeth Delozanne et Françoise Le Calvez de l'Université Pierre et Marie Curie de Paris.

6. Bibliographie

Abt C., *Serious Games*, University Press of America, 1970.

ACM/IEEE (ed.), *Computing Curricula 2005, The Overview Report*, IEEE Computer Society Press. and ACM Press., New York, 2005.

Chen W.-K., Cheng Y. C., « Teaching Object-Oriented Programming Laboratory With Computer Game Programming », *Education, IEEE Transactions on*, vol. 50, n° 3, p. 197-203, August, 2007.

Cobb P., Confrey J., DiSessa A., Lehrer R., Schauble L., « Design Experiments in Educational Research », *Educational Researcher*, vol. 32, n° 1, p. 9-13, January, 2003.

Crenshaw T. L., Chambers E. W., Metcalf H., Thakkar U., « A case study of retention practices at the University of Illinois at Urbana-Champaign », *39th ACM Technical Symposium on Computer Science Education*, vol. 40, n° 1, p. 412-416, 2008.

Fincher S., Petre M., Clancy M., Clear T., Guzdial M., Hundhausen C. D., McCracken W. M., Rist R. S., Stasko J. T., *Computer Science Education Research*, Taylor & Francis The Netherlands, Lisse, chapter 1, p. 1-8, 2004.

Gestwicki P., Sun F.-S., « Teaching Design Patterns Through Computer Game Development », *ACM Journal on Educational Resources in Computing*, vol. 8, n° 1, p. 1-22, 2008.

Ginat D., « On Novice Loop Boundaries and Range Conceptions », *Computer Science Education*, vol. 14, n° 3, p. 165-181, 2004.

Greitzer F. L., Kuchar O. A., Huston K., « Cognitive science implications for enhancing training effectiveness in a serious gaming context », *J. Educ. Resour. Comput.*, vol. 7, n° 3, p. 2, 2007.

Hartness K., « Robocode : using games to teach artificial intelligence », *J. Comput. Small Coll.*, vol. 19, n° 4, p. 287-291, 2004.

Johnson W. L., Vilhjalmsson H., Marsella S., « Serious Games for Language Learning : How Much Game, How Much AI ? », *12th International Conference on Artificial Intelligence in Education (AIED)*, Amsterdam, The Netherlands, july, 2005.

Kelleher C., « Alice and The Sims : the story from the Alice side of the fence », *The Annual Serious Games Summit DC Washington*, DC, USA, October 30–31,, 2006.

Kelleher C., Cosgrove D., Culyba D., Forlines C., Pratt J., Pausch R., « Alice2 : Programming without Syntax Errors », *15th annual symposium on the User Interface Software and Technology*, Paris, France, October, 2002.

Klopfer E., Yoon S., « Developing Games and Simulations for Today and Tomorrow's Tech Savvy Youth », *TechTrends : Linking Research and Practice to Improve Learning*, vol. 49, n° 3, p. 33-41, 2005.

Leutenegger S., Edgington J., « A games First Approach to Teaching Introductory Programming », *SIGCSE '07 : Proceedings of the 38th SIGCSE technical symposium on Computer science education*, vol. 39, n° 1, p. 115-118, March, 2007.

Maloney J., Burd L., Kafai Y., Rusk N., Silverman B., Resnick M., « Scratch : A Sneak Preview », *2nd International Conference on Creating Connecting, and Collaborating through Computing*, Keihanna-Plaza, Kyoto, Japan, p. 104-109, January, 2004.

Michaud L., Alvarez J., Serious games : Advergaming, edugaming, training..., Technical report, IDATE Consulting & Research, 2008.

Muratet M., Torguet P., Jessel J.-P., Viallet F., « Towards a serious game to help students learn computer programming », *Int. J. Comput. Games Technol.*, vol. 2009, p. 1-12, 2009.

Seppälä O., Malmi L., Korhonen A., « Observations on student misconceptions - A case study of the Build Heap Algorithm », *Computer Science Education*, vol. 16, n° 3, p. 241-255, 2006.

Siang A., Rao R. K., « Theories of learning : a computer game perspective », *Multimedia Software Engineering, 2003. Proceedings. Fifth International Symposium on*, p. 239-245, December, 2003.

Soloway E., Bonar J., Ehrlich K., « Cognitive Strategies and Looping Constructs : An Empirical Study », *Communications of the ACM*, vol. 26, n° 11, p. 853-860, 1983.

Stamm S., « Mixed nuts : atypical classroom techniques for computer science courses », *Crossroads The ACM Student Magazine*, 2007.

Stevenson D. E., Wagner P. J., « Developing real-world programming assignments for CS1 », *ITICSE '06 : Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, Bologna, Italy, p. 158-162, June, 2006.

Zyda M., « From Visual Simulation to Virtual Reality to Games », *IEEE Computer*, vol. 38, n° 9, p. 25-32, 2005.

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com

ANNEXE POUR LE SERVICE FABRICATION
A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :
RSTI - RIA. Volume X – n˚X/20XX

2. AUTEURS :
Mathieu Muratet[*,***] *— Patrice Torguet*[*,***] *— Fabienne Viallet*[**,***] *— Jean-Pierre Jessel*[*,***]

3. TITRE DE L'ARTICLE :
Évaluation d'un jeu sérieux pour l'apprentissage de la programmation

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
Un jeu sérieux pour la programmation

5. DATE DE CETTE VERSION :
25 février 2011

6. COORDONNÉES DES AUTEURS :

– adresse postale :

[*] IRIT, équipe VORTEX

{Mathieu.Muratet, Patrice.Torguet, Jean-Pierre.Jessel}@irit.fr

[**] CREFI-T, équipe DiDiST

fabienne.viallet@iut-tlse3.fr

[***] Université Paul Sabatier, 118 Route de Narbonne F-31062 Toulouse cedex 9

– téléphone : 00 00 00 00 00

– télécopie : 00 00 00 00 00

– e-mail : guillaume.laurent@ens2m.fr

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
LaTeX, avec le fichier de style `article-hermes.cls`,
version 1.23 du 17/11/2005.

8. FORMULAIRE DE COPYRIGHT :
Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
`http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com